Simultaneous Localization and Surveying with Multiple Agents

Sam T. Roweis and Ruslan R. Salakhutdinov

Department of Computer Science, University of Toronto
Toronto, Ontario, M5S3G4, CANADA,
{roweis,rsalakhu}@cs.toronto.edu

Abstract. We apply a constrained Hidden Markov Model architecture to the problem of simultaneous localization and surveying from sensor logs of mobile agents navigating in unknown environments. We show the solution of this problem for the case of one robot and extend our model to the more interesting case of multiple agents, that interact with each other through proximity sensors. Since exact learning in this case becomes exponentially expensive, we develop an approximate method for inference using loopy belief propagation and apply it to the localization and surveying problem with multiple interacting robots. In support of our analysis, we report experimental results showing that with the same amount of data, approximate learning with the interaction signals outperforms exact learning ignoring interactions.

1 Introduction

In the following, we study the problem of analyzing sensor logs created by mobile agents navigating in unknown environments. We assume that the environment is static, so that any variation in the sensors is caused by the movement of the agents in the world. Our goal is to develop algorithms for localization when the environment is known and also for simultaneous localization and identification of the environment, which we dub "surveying". We also consider the situation of multiple agents that have limited but nontrivial interaction as they explore. All of these problems can be cast as statistical estimation computations and approached using techniques of probabilistic inference.

Simultaneous localization and surveying (SLAS) is distinct from the well known simultaneous localization and mapping (SLAM) problem. In SLAS we are not trying to learn the occupancy grid of a world, rather we are trying to learn the values that various sensors (e.g. altitude, temperature, light level, beacon signals) take on as a function of position in the unknown environment. Furthermore, we cannot control the movement of the agent, we can only analyze the sensor logs recorded as it traverses the world. This task is motivated by agents (for example mobile planetary rovers) which generally operate in open spaces, collect temporal histories of multiple sensors, and cannot rely on the odometry of self-locomotion (e.g. because they are navigating extremely rough terrain or not using conventional wheels).

R. Murray-Smith, R. Shorten (Eds.): Switching and Learning, LNCS 3355, pp. 313–332, 2005.

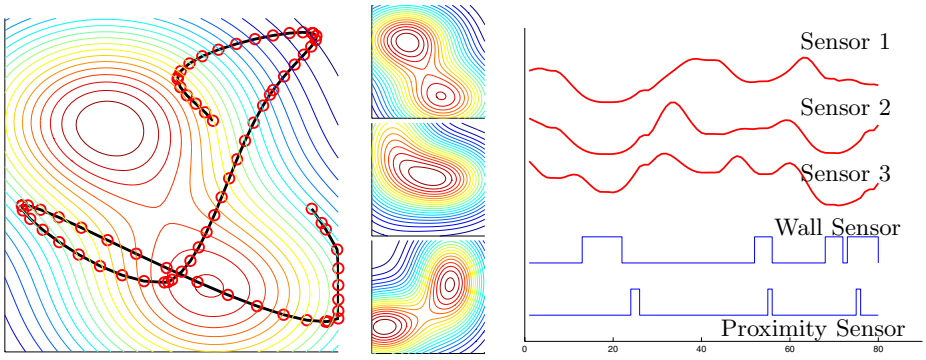

Fig. 1. Typical noisy sensor readings seen by a single agent. The first three sensors are continuous valued sensors; the fourth is a wall-detecting binary sensor. The proximity sensor indicates whether an agent is in the close proximity with other agents.

Figure 1 shows the typical input to a single agent (robot) in the scenario we are studying. Each robot moves through the environment under the control of an external navigation algorithm that we cannot influence. As it proceeds, it logs readings from multiple noisy sensors, some of which may be smoothly varying functions of its position in the world and others of which may be intermittent or discontinuous. No odometry or other information about navigational control signals (either intended or realized) is available to the agent. When there are multiple agents navigating the environment at the same time, and they interact in some way, the problem becomes much more rich. (Without interaction, the pooled data from all agents is almost identical to one longer data stream from a single agent.) In the following, we consider a very simple type of interaction: we assume that agents can detect each other when they are in close proximity. The most general problem we wish to solve is to simultaneously discover the trajectories taken by each robot (localization) and to learn the values of each sensor variable across the environment (surveying).

We approach the localization task as one of state inference in a probabilistic model and the surveying task as one of unsupervised learning of the model parameters. In the inference problem, the location of each agent over time is treated as a hidden variable that is to be inferred, conditioned on the observed sensor readings (and possibly proximity interactions with other agents). Inference on its own assumes we are given the model survey parameters, which specify the distributions of sensor values as a function of position across the space. The learning problem addresses the estimation of these model parameters (sensor map) given the agent locations over time. The two problems are interconnected, since to localize, an agent must know the sensor map, but to contribute to the learning of this map, it must have an estimate of location in the unknown environment.

One very effective way of tackling these problems is to discretize the world into small spatial cells and to identify each such cell with a discrete state in a

dynamic Bayesian network such as a Hidden Markov Model (HMM) or a series of coupled HMMs. When the state is low-dimensional (e.g. containing only a two dimensional position and no orientation information), this discretization is expensive but possible. However, in higher dimensions the number of cells required scales prohibitively. In fact, this discretization is simultaneously the source of the algorithm's power and its greatest computational challenge.

Once discretized, we can treat the state as a latent variable and apply standard statistical learning methods for discrete state models. The key insight is that by identifying each state in the hidden Markov model with some small spatial region of the continuous world space, it is possible to naturally define "neighbouring" states as those which correspond to connected regions in the underlying space. The transition matrix of the HMM can then be *constrained* to allow transitions only between neighbors; this means that all valid state sequences correspond to connected paths in the continuous space. The transition matrix does not need to be explicitly stored or learned, it is merely computed by a function that respects the state topology; the remaining parameters of the model scale only linearly with the number of states[12].

For a single agent, or multiple non-interacting agents, the learning and inference algorithms are identical to those for standard HMMs trained on multiple observation sequences, except that the transition matrix of the HMM is fixed by the spatial topology of the problem and is not updated during learning.

Another way to address the problem is to represent the state of each agent using a distribution over the continuous domain of the random variables. Because the distribution on these variables can be highly complex and multimodal, the most natural representation is to store a set of (weighted) samples as an approximation of the distribution. The greatest advantage of this approach is that it does not require discretization of the entire state space and thus does not scale up prohibitively with the number of spatial cells. This general approach is known as "particle filtering", and the central technical challenge is how to update the particles to efficiently represent the state posterior.

In this work, we take the very simple approach of searching only for the mode of the true posterior, and use a single particle at each time to represent the state of the agent. Optimizing this set of particles to find the maximum a posteriori state trajectory is a very difficult search problem. We use the *embedded hidden Markov model* architecture discussed by [9, 10] as our search engine. (Although originally it was proposed as a more sophisticated method for drawing samples from the exact posterior.) The search begins by forming a pool of possible candidate states at each time, candidates, representing the agent's possible locations in the unknown environment. Given the pool, we can restrict the agent's state to only those values represented by candidates at each time, and thus define an "embedded HMM", whose discrete states are the indices within each pool time. By performing efficient Viterbi-style decoding, we can find the most probable trajectory of the agent, constrained to pass only through the existing pool states. If we always include the current best estimate of the state into the pool of candidates at each time, we are guaranteed to either find a new

and improved (more probable) state sequence or to retain the same trajectory we currently have. After the dynamic programming, we create a new pool by randomly sampling states at each time in the vicinity of the current best state estimate.

The most difficult and interesting case we explore in this work is that of multiple agents that navigate through the environment simultaneously and interact in some way with each other. In this case, exact inference requires estimating the joint state of all agents, and quickly becomes exponentially expensive because the effective state space is the product of the state spaces of the individual robots. For the case of discretized world, we develop an approximate but efficient and accurate method for solving this multiple-inference problem using belief propagation (BP). We apply our BP algorithm to the localization and surveying problem with multiple interacting robots and show that approximate learning using multiple agents with interaction signals outperforms exact learning using the same amount of data but ignoring the interaction signals that make the problem more difficult.

2 Localization with a Single Robot

In this section we develop techniques to solve the localization problem for a single robot, which navigates in an unknown environment and records some observations (continuous or binary) from its sensors. We denote the observation at (discrete) time t from sensor c by y_{ct}, the entire vector of sensor readings at time t by $\boldsymbol{y}_t$, and the unknown state (location) of the robot at time t by s_t.

The probabilistic graphical model that relates the empirical observation sequence $Y = \{\boldsymbol{y}_1, ..., \boldsymbol{y}_T\}$ to the hidden state sequence $S = \{s_1, ...s_T\}$ is shown in figure 2. This model specifies a factorization of the joint distribution between the trajectory and the observation sequence as:

$$p(Y, S) = \prod_t p(s_t|s_{t-1})p(\boldsymbol{y}_t|s_t) \tag{1}$$

Our goal in localization is to find the optimal trajectory given a sequence of observations:

$$\arg\max_S \log P(S|Y) = \arg\max_S \sum_t \left[\log P(s_t|s_{t-1}) + \log P(\boldsymbol{y}_y|s_t)\right] \tag{2}$$

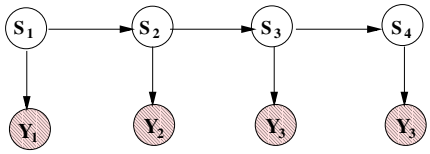

Fig. 2. The graphical model that displays the relationship between the state sequence $s_1, ..., s_4$, and the empirical observation sequence $\boldsymbol{y}_1, ..., \boldsymbol{y}_4$. for $t = 1, .., 4$ time steps

Later, when we are interested in identifying the parameters (survey maps) of the unknown world, our goal will be to maximize the likelihood of the observations given the (survey) parameters, integrating (summing) over all possible paths the robot could have taken:

$$\max_{\theta} \log P(Y|\theta) = \log \sum_{S} \sum_{t} \left[\log P(s_t|s_{t-1}, \theta) + \log P(\boldsymbol{y}_y|s_t, \theta)\right] \tag{3}$$

As a byproduct of this parameter learning we will end up inferring the marginal posterior of the robot's position at each time, thus also performing a form of average localization. (It is also possible to use our inference about the single most probable (Viterbi) path of each agent to do a form of MAP learning, although in the discussion below we focus on maximum likelihood estimation which sums over all possible paths.)

2.1 Discretizing the World

Our first approach to solving the localization problem is to discretize the world into small spatial cells. By identifying each state in a hidden Markov model with some small spatial region of a continuous space, it is possible to naturally define neighboring states as those which correspond to connected regions in the underlying space, which leads us to the constrained HMM architecture [12]. The transition matrix of the HMM is precomputed to allow transitions only between neighbors; this means that all valid state sequences correspond to connected paths in the continuous space. The transition matrix does not need to be explicitly stored or learned, it is merely computed by a function that respects the state topology; the remaining parameters of the model scale only linearly with the number of states. Given these constraints, localization reduces to inference in this sparsely connected HMM, and can be solved using the well known Viterbi decoder.

To represent the world map of each continuous sensor c, we assume a conditional Gaussian model given the index of the discrete state: $P(y_{ct}|s_t = i) = \mathcal{N}(y_{ct}; m_i^c, \sigma_i^c)$. For binary sensors d we assume a simple Bernoulli model: $P(y_{dt} = 1|s_t = i) = m_i^d$. We assume that the noise in the sensor observations is uncorrelated from sensor to sensor and also over time (white).

An example of using this approach for localization (given knowledge of the sensor maps) is presented in figure 3 (left panel). Note that inference using this approach will always be only approximate due to the discretization error.

The discretization of the continuous state space can be very expensive and presents the model's greatest computational challenge. In higher dimensions or for very large areas requiring fine spatial resolution to reduce the discretization error, the number of cells required scales prohibitively. To alleviate this problem, we can represent the state of the robot using continuous random variables, as discussed in the following subsection. However, the optimization (search) problem of finding the best path given a history of sensor readings become extremely difficult.

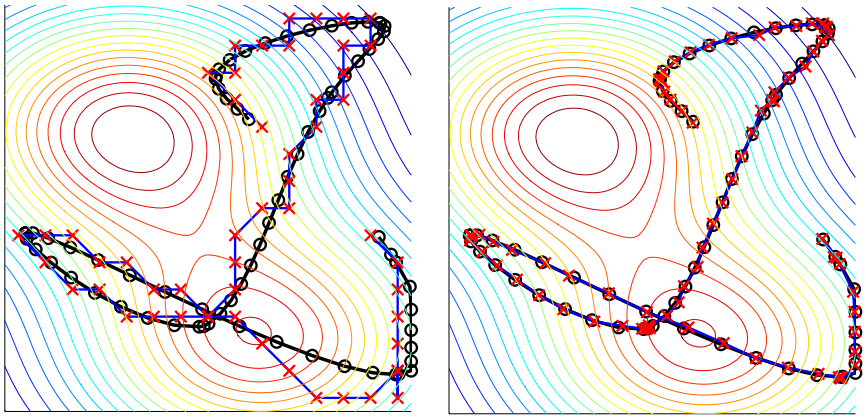

Fig. 3. Display of the discretized approach (left panel) and embedded HMM approach (right panel) to the robot localization problem, given the true map of sensor readings. The "o"s indicate the time slices when the true observation was taken, and "x"s indicate the inferred path taken by the robot.

2.2 Continuous Representation

To avoid the exponential cost of discretization as well as the unavoidable estimation error it imposes on localization, we can attempt to represent the trajectories of each agent using truly continuous random variables. However, two major difficulties arise when taking this approach: first, how should we represent a joint distribution over states at all times and second, how should we optimize this distribution given the sensor readings and the maps? These issues are central to the study of inference in all nonlinear dynamical systems and various approximation solutions have been proposed. Here, we follow the common programme of representing the distribution using sample trajectories, this approach is often called "particle filtering" or "Monte Carlo filtering".

We consider the robot's continuous trajectory to be a sequence of unknown positions $L = \{\boldsymbol{\ell}_1, ..., \boldsymbol{\ell}_T\}$ and our goal is to infer something about L given the empirical observation sequence $Y = \{\boldsymbol{y}_1, ..., \boldsymbol{y}_T\}$ and the known observation functions $p(y_c|\boldsymbol{\ell})$ for each sensor. In the most ambitious setting, we could attempt to infer the full joint posterior over L given Y, but as a first step we consider only finding the mode of L, i.e. the most likely trajectory given the observations.

For this search, we apply the embedded hidden Markov model[9, 10] as a simple optimization technique to solve the localization problem without the need to discretize the world. The optimization starts with some initial guess $\hat{L} = \{\hat{\boldsymbol{\ell}}_1, \ldots, \hat{\boldsymbol{\ell}}_T\}$ of the state trajectory, perhaps from a very coarse discretization or other approximation. At each step of the search, we create a pool of candidate states at each time. The pool contains K members, representing possible locations of the agent at each time step t. The candidate states within each pool are generated according to a proposal distribution $Q_t(\hat{\boldsymbol{\ell}}_t)$. In our experiments we

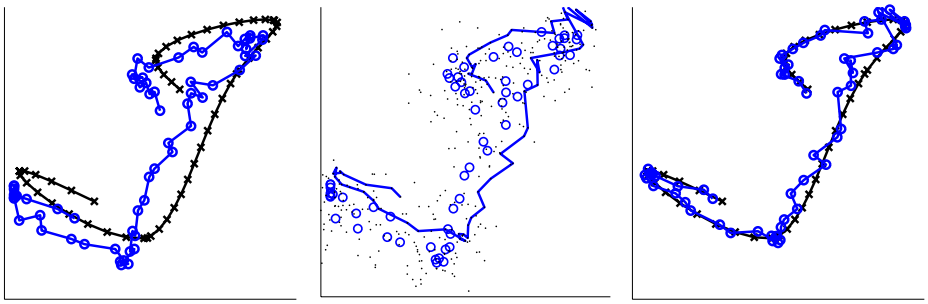

Fig. 4. Display of one iteration of the embedded HMM optimizer. Left panel shows part of the robot's true and currently inferred trajectory, where the "x"s indicate the time slices when the true observation was taken, and "o"s indicate the currently inferred path taken by the robot. The middle panel displays the candidate states (dots) within each pool along with the more probable path after performing Viterbi decoding. The right panel shows the true along with the newly inferred trajectory.

have used simple Gaussian proposals with the mean set to the current guess $\hat{\boldsymbol{\ell}}_t$ and a fixed (isotropic) covariance equal at all time steps: $Q_t = \mathcal{N}(\hat{\boldsymbol{\ell}}_t, \sigma I)$. This encourages the points placed into the pool to represent plausible alternatives to the current guess $\hat{\boldsymbol{\ell}}_t$ about the state at time t. Crucially, we also include the current guess in the pool.

If we now constrain our search to only consider locations represented by pool states, the collection of pools across time define an "embedded HMM", whose $K+1$ states are the indices within each pool. By performing Viterbi decoding on this resulting embedded HMM, we can efficiently search through an exponential number of trajectories to find the best one. As well, we are guaranteed to always find a trajectory that increases $P(\hat{L}|Y)$ or leaves it the same, since the current guess at each time is always included in the pool. The optimization is repeated until several steps have passed with no change to our best trajectory estimate. The complete inference algorithm using embedded HMM is given below:

Localization Algorithm using Embedded HMM:

- Initialize $\hat{L} = \{\hat{\boldsymbol{\ell}}_1, ..., \hat{\boldsymbol{\ell}}_T\}$
- Repeat until a better trajectory cannot be found:
  - Form pools of candidate states for each time step t by:
    - Including the current state $\hat{\boldsymbol{\ell}}_t$ at time t into the pool for time t
    - Sampling K other candidate states for each pool from $Q_t(\hat{\boldsymbol{\ell}}_t)$
  - Define an embedded HMM, whose $K+1$ states are the indices within each pool
  - Perform Viterbi decoding to select the new trajectory $\hat{L}_{new}$ through the pool states that increases $P(L|Y)$ or leaves it the same.
  - Set $\hat{L} = \hat{L}_{new}$

Of course, the initial trajectory estimate greatly affects the quality of the final path returned by the search. If we were to randomly initializing the initial

guess, the algorithm would often get stuck in a poor local optimum. To avoid this problem, we first subsample time by forming blocks of contiguous sequential empirical observations (each block containing N observations). The embedded HMM optimizer is then applied as above, using only one pool per block to find a coarse estimate of the trajectory, effectively subsampled by a factor of N. This will help our algorithm to roughly locate the parts of the sensor maps that best explain each block of empirical observations. Once convergence of the search has been achieved at the coarse level, the resulting estimate is then used as an initial trajectory input into a finer level of grouping, and so on until the full resolution of the problem is reached.

An example of using embedded HMM approach for localization (given the sensor maps of the world) is presented in figure 3 (right panel). First, the embedded HMM optimizer is applied for the blocks of N=20 sensor observations, then for blocks of length N=10 (using the final trajectory from N=20 as the initialization); then N=5 and finally at the full resolution of the problem.

From the results in figure 3, it can be seen that the embedded HMM is capable of achieving arbitrarily good accuracy in reconstructing the agent trajectories, while the discrete state constrained HMM approach is ultimately limited by the discretization error of the grid. However, the discrete state representation allows us to easily represent and compute a distribution over state trajectories, for example by computing the marginal uncertainty of state occupations at each time. Such a distribution will be useful for learning the parameters of an unknown world, as discussed below.

3 Simultaneous Localization and Surveying

We now turn our attention to the more ambitious problem of analyzing sensor logs from agents operating in unknown environments. Here, our goal is to simultaneously learn (identify) the sensor maps describing the world and localize the agents by estimating their trajectories. We have dubbed this problem Simultaneous Localization and Surveying (SLAS), in contrast to the related problem of Simultaneous Localization and Mapping (SLAM).

Of course, the SLAS problem is only solvable up to certain identifiability limitations. The absolute rotation and reflection of the true world map can never be recovered since rotating or flipping the world and simultaneously rotating and flipping our trajectory estimates will result in identical likelihood of the observed sensor logs. Similarly, the scale of the world cannot be recovered unless we have prior knowledge about the agents velocities. However, up to these degeneracies, the problem is still worth investigating, as we show below.

3.1 Single Agent

For the case of discretized world, and a single agent, we can use the well known EM algorithm for learning the parameters of the effective Hidden Markov Model being used to model the world. In the case of HMMs, the EM algorithm is known

as Baum-Welch, and the associated equations are very well known. In our case, the HMM is highly constrained, which makes learning much easier since we do not need to estimate the state transition matrix. This matrix is fixed by the spatial topology of states, allowing transitions only between neighbors, and is not updated during learning. In fact, the sparsity of the spatial topology results in very efficient inference and learning, since very few entries of the transition matrix are nonzero and sums need be performed only over these. The learning equations for the output distributions of each state, which represent the sensor maps, are the same as for a regular HMM. Inference is performed using the standard forward-backwards recursions, which are a special case of belief propagation applied to the graphical model of the HMM.

An example of simultaneously learning the survey parameters and estimating the agent's trajectory is shown in the second row of figure 7.

When the sensor maps are known a priori, localization can be performed optimally, and given only a small amount of data it is usually possible to discover the trajectory of the agent quite accurately. However, when simultaneously learning the maps and performing localization the problem is much harder. With small amounts of data (short trajectories), parts of the environment map that have similar patterns of sensor readings are difficult to distinguish from one another. Therefore a single robot may have difficulty accurately localizing itself.

The problem can be alleviated by having multiple robots which explore the environment simultaneously and interact with each other, for example through proximity sensors. (If the robots were not interacting, the problem would be exactly equivalent to that of a single robot who explored the environment on several independent excursions.)

In the interacting agents case, exact learning requires inferring the joint state of all robots, and quickly becomes exponentially expensive because the effective state space of the HMM is the product of the state spaces of the individual robots. In the next section we develop an approximate but efficient method for inference in this case using belief propagation and apply it to the SLAS problem with multiple interacting robots.

3.2 Multiple Interacting Agents

Consider a scenario in which multiple robots explore the environment simultaneously and interact with each other by communicating signals between them. In what follows, we only consider a very simple form of interaction: the robots are equipped with proximity sensors which notify them when they are near another robot. The proximity signal includes the identity of the other agent encountered, but not a relative heading.

The new graphical model relating empirical observation sequences, interaction signals, and hidden state sequences is shown in figure 5 for two robots and four time steps. This model is very similar to the Factorial HMM[4], except that there are both private outputs from each chain (in our case the sensor readings of each robot) as well as shared outputs (in this case the interaction signals).

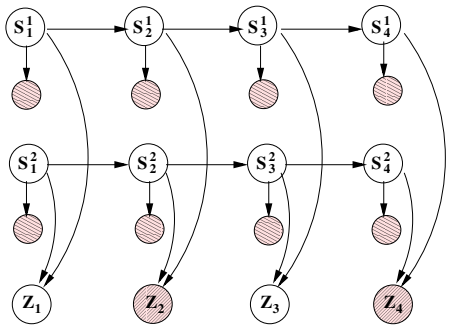

Fig. 5. The graphical model, that displays the relationship between state chains of two agents $S^1 = \{s_1, ..., s_4\}$, $S^2 = \{s_1, ..., s_4\}$, their empirical observations, and the interaction signals $z_1, ..., z_4$ for 4 time steps. The interaction signal z_t is shaded if it is on, which forces the state chains to become coupled.

Note that, even though the state chains are a priori independent, once we condition on the interaction evidence, the chains become coupled. This makes inference much more difficult (and similarly learning, which requires inference), since we can no longer run the simple forward-backward recursions independently on each robot's chain. Of course, the factorial representation of the coupled HMM can always be transformed into a regular HMM, whose effective state space is the Cartesian product of the state spaces of the individual robots. However, inference in this "flattened" model requires working in the joint state space of all robots and quickly becomes exponentially expensive.

Several approaches can be taken to tackle this challenging inference computation. Stochastic sampling algorithms, usually based on importance sampling or Markov Chain Monte Carlo[5] can provide randomized (but often unbiased) estimates of state occupation statistics. One can also employ structured variational approximations to the posterior over hidden states, similar to the ones discussed in [4], and proceed to optimize a lower bound on the likelihood.

Another alternative is to apply a class of approximate (i.e. biased) inference algorithms that are based on belief propagation [11]. Of course, for the uncoupled HMM, the standard HMM inference algorithms are exactly equivalent to belief propagation on a particular junction tree constructed from the original graphical model. For the coupled HMM, we derive below an approximate method for inference which uses loopy belief propagation (LBP) on the equivalent junction tree. Loopy BP passes messages exactly as in regular belief propagation, ignoring the cycles in the graph. The messages are passed according to a predetermined schedule and beliefs are updated in the standard way. Although approximate, LBP has proved to be very successful in practice in many other domains[3, 2]

In theory, LBP runs the risk of "overcounting" information, and thus may not converge or may converge to the wrong answer. However, for our particular problem we find the algorithm very suitable. Indeed, it has been observed in

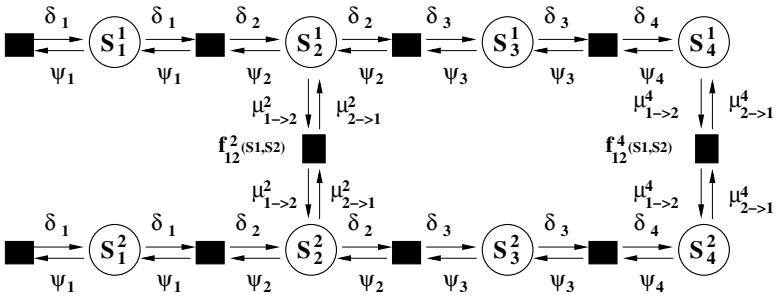

Fig. 6. Factor graph representation of the original Bayes net along with message propagation.

practice that if the original graph does not contain dense loops, LBP usually converges and produces very good approximation. In terms of our setup, if the interaction signals are infrequent (robots do not meet each other very often), then our graphical model will have exactly this characteristic of large, loosely coupled cycles; which is well matched to loopy propagation. In the experimental section we confirm this intuition: loopy BP on our graphs almost always converges to sensible beliefs; and when it is used as the inference step in learning the resulting maps accurately survey the true world sensor maps.

4 Localization with Multiple Agents: Loopy Propagation

The multiple agent localization problem focuses on inferring the most probable trajectories (state sequences) of each agent conditioned on the observed sensor readings of all agents, and the proximity interactions between agents.

To solve this problem, we first pursue only the mode of the joint distribution over agent trajectories. That is, we try to find the single set of trajectories (one for each agent) that simultaneously explain the interaction signals and the sensor logs of each agent as well as possible.

Our approach is to develop below the variant of LBP known as max-product loopy belief propagation [13]. This is exactly the equivalent of Viterbi decoding for coupled HMMs. To derive the necessary equations, we first convert the original Bayes net (fig. 5) to its factor graph (fig. 6) representation [7]. The factor graph contains both variable nodes and factor nodes and is bipartite: edges exist only between variables and factors. Messages flow only from factor nodes to variables and back, but never between factors or between variables. Once a variable node has received messages from all other neighboring factor nodes it takes the product of these messages and delivers it to the destination factor node. The message that a factor node f sends to a variable node x is the maximum over all quantities not present in x of the product of all the incoming messages to f from other neighboring variable nodes y. The incoming messages to f are also multiplied by the potential function defined at f before the max is taken:

$$\mu_{x\to f}(x) = \prod_{h\in neigh(x)\setminus\{f\}} \mu_{h\to x}(x) \tag{4}$$

$$\mu_{f\to x}(x) = \max_{X\setminus x} f(X) \prod_{y\in neigh(f)\setminus x} \mu_{y\to f}(x) \tag{5}$$

(Note that if we ignore the proximity interactions, the max-product algorithm reduces to performing Viterbi decoding separately on each robot's state chain given its sensor observations, i.e. treating it as a regular HMM.) All that remains is to specify a message passing schedule. In principle, one could apply any schedule including running the above updates in parallel. We choose a message passing schedule in which we cycle through chains, passing messages across time for one chain, then from that chain to all others (using the proximity potentials), and then across time in the next chain. In effect, we are performing Viterbi on each chain, taking into account both its observations and the effect of the messages it receives from other chains (which appear as pseudo-observations). This schedule is sometimes known as "chainwise Viterbi".

To formalize the algorithm, we must define the proximity (interaction) potential function $f_{qr}(s^q, s^r)$ that couples robots r and q. In general, one would like to account for noisy interaction signals which are functions of the true separation of the robots. In many applications, the proximity sensors have an extremely low false positive rate, and a moderately low false negative rate. To make inference efficient, and keep the loops in our graphical model as large as possible, we approximate the proximity potential using the assumption that the false positive rate is zero. Thus, at time slice t if a proximity signal between q and r is detected, we set

$$f^t_{qr}(s^q_t, s^r_t) = N(s^q_t, s^r_t)$$

$$\text{where } N(s^q_t, s^r_t) = \begin{cases} 1 \text{ if } s^q_t \text{ and } s^r_t \text{ are neighbouring states} \\ 0 \text{ otherwise} \end{cases}$$

Otherwise, if a proximity signal between q and r sensor is not observed ($p^t_{qr} = 0$), we define $f_{qr}(s^q, s^r)$ to be a constant.

Note that this definition implies that when a proximity signal is observed, both robots *must* be in neighbouring (or identical) states, which according to our problem corresponds to neighbouring or identical discretized spatial cells. (Ideally, the definition of the potential function could be more sophisticated, but this would increase the complexity of inference.) The message at time slice t that a variable node s^q_t node sends to the factor node f^t_{qr} takes the form:

$$\mu^t_{q\to f_{qr}} = \prod_{h\in neigh(s^q_t)\setminus\{f^t_{qr}\}} \mu_{h\to s^q_t} \tag{6}$$

Then the message that a factor node f^t_{qr} sends to the variable node s^r_t is:

$$\mu^t_{f_{qr}\to r} = \max_{s^q_t} f^t_{qr}(s^q_t, s^r_t) \prod_{y\in neigh(f_{qr})\setminus s^r_t} \mu^t_{y\to f_{qr}} \tag{7}$$

Initially, all messages are set to one.

Max-Product Loopy Belief Propagation executed by each robot r:

• Define: $\delta_t^r(i) = \max_{s_1^r,...,s_{t-1}^r} p(s_1^r, ..., s_t^r = i, \boldsymbol{y}_1^r, ..., \boldsymbol{y}_t^r|\Theta)$;
. $\psi_t^r(i) = \max_{s_T^r,...,s_{t+1}^r} p(s_T^r, ..., s_t^r = i, \boldsymbol{y}_{t+1}^r, ..., \boldsymbol{y}_T^r|\Theta)$

– Initialize $\delta_1^r(i) = \pi_i^r p(\boldsymbol{y}_1^r|s_1^r = i), \quad \psi_T^r(i) = 1 \quad 1 \le i \le N,$
with N being the number of states.

– Induction for $1 \le j \le N$ and $1 \le i \le N$:

$$\delta_{t+1}^r(j) = \left[\max_i \delta_t^r(i) T_{ij} \prod_{q:\text{proximity}} \mu_{q\to r}^t(i)\right] p(\boldsymbol{y}_{t+1}^r|s_t^r = j); \quad 1 \le t \le T-1$$

$$\psi_t^r(i) = \max_j T_{ij} p(\boldsymbol{y}_{t+1}^r|s_t^r = j)\psi_{t+1}^r(j) \prod_{q:\text{proximity}} \mu_{q\to r}^{t+1}(j) \quad t = T-1, ..., 1$$

– Termination: Compute local beliefs and a new set of messages.

$$\gamma_t^r(i) = \frac{\delta_t^r(i)\psi_t^r(i)\prod_q \mu_{q\to r}^t(i)}{\sum_j \delta_t^r(j)\psi_t^r(j)\prod_q \mu_{q\to r}^t(j)}$$

For proximity signals, where Z is the normalization constant

$$\mu_{r\to q}^t(i) = \frac{1}{Z}\max_{s_t^r}\left[f_{rq}^t(s_t^r, s_t^q = i)\delta_t^r(i)\psi_t^r(i)\prod_{p\neq q}\mu_{p\to r}^t(i)\right] \quad \forall q$$

The final max-product algorithm run by each robot is given above. Our message passing schedule is quite simple: we iterate through robots $r = 1, 2, ..., R$ sequentially, running the above max-product algorithm (which includes the effect of all incoming messages). Afterwords, the algorithm computes local beliefs $\gamma_t^r(i)$ and all outgoing messages $\mu_{r\to q}^t$ are sent to all other agents. We monitor the convergence of this LBP by the absolute difference between successive local beliefs and continue passing messages until these stabilize which typically takes 4-5 iterations We have also experimented with running the the above message updates in parallel, and obtained exactly the same results, although with slightly slower convergence.

5 Multi-SLAS: Learning with Multiple Agents

The final problem we discuss is the most difficult: simultaneous localization and surveying using sensor logs from multiple interacting robots. Our approach to this problem is to employ a loopy belief propagation (LBP) method very similar to the one from the previous section as the inference engine, and to alternate between approximate inference using this new version of LBP and parameter (sensor map) estimation based on the results of this inference.

To develop the new LBP equations, which we will use for learning, we focus not on the mode of the distribution but on its marginals. In other words, we use sum-product instead of max-product in an attempt to integrate over all possible paths that the multiple agents could have taken. This is analogous to the forward-backward (alpha-beta) procedure for a single HMM but now in the case of our coupled HMM chains. Once again, we convert the original Bayes net

to its factor graph representation. As before, once a variable node has received messages from all other neighboring factor nodes it takes the product of these messages and delivers it to the destination factor node. However, in contrast to the max-product algorithm, the message that a factor node sends to a variable node is the *marginalized* product of all the incoming messages from its other neighboring variable nodes, multiplied by its current potential function:

$$\mu_{x\to f}(x) = \prod_{h\in neigh(x)\setminus\{f\}} \mu_{h\to x}(x) \tag{8}$$

$$\mu_{f\to x}(x) = \sum_{X\setminus x} f(X) \prod_{y\in neigh(f)\setminus x} \mu_{y\to f}(x) \tag{9}$$

Such algorithms are thus often termed "sum-product" algorithms. We employ a message passing schedule identical to the one used for max-product loopy propagation. Using the same potential functions as before,the message at time slice t that a variable node s_t^q node sends to the factor node f_{qr}^t takes the form:

$$\mu_{q\to f_{qr}}^t = \prod_{h\in neigh(s_t^q)\setminus\{f_{qr}^t\}} \mu_{h\to s_t^q} \tag{10}$$

Then the message that a factor node f_{qr}^t sends to the variable node s_t^r is:

$$\mu_{f_{qr}\to r}^t = \sum_{s_t^q} f_{qr}^t(s_t^q, s_t^r) \prod_{y\in neigh(f_{qr})\setminus s_t^r} \mu_{y\to f_{qr}}^t \tag{11}$$

Sum-Product Loopy Belief Propagation executed by each robot r:

• Define: $\alpha_t^r(i) = p(\boldsymbol{y}_1^r, ..., \boldsymbol{y}_t^r, s_t^r = i|\Theta)$; $\beta_t^r(i) = p(\boldsymbol{y}_{t+1}^r, ..., \boldsymbol{y}_T^r, s_t^r = i|\Theta)$

. $\gamma_t^r(i) = p(s_t^r = i|Y, \Theta)$

– Initialize $\alpha_1^r(i) = \pi_i^r p(\boldsymbol{y}_1^r|s_1^r = i)$, $\beta_T^r(i) = 1$ $1 \le i \le N$, with N being the number of states.

– Induction for $1 \le j \le N$ and $1 \le i \le N$:

$$\alpha_{t+1}^r(j) = \Big[\sum_i \alpha_t^r(i) T_{ij} \prod_{q:\text{proximity}} \mu_{q\to r}^t(i)\Big] p(\boldsymbol{y}_{t+1}^r|s_t^r = j); \quad 1 \le t \le T-1$$

$$\beta_t^r(i) = \sum_j T_{ij} p(\boldsymbol{y}_{t+1}^r|s_t^r = j)\beta_{t+1}^r(j) \prod_{q:\text{proximity}} \mu_{q\to r}^{t+1}(j) \quad t = T-1, ..., 1$$

– Termination: Compute marginal beliefs and a new set of messages

$$\gamma_t^r(i) = \frac{\alpha_t^r(i)\beta_t^r(i)\prod_q \mu_{q\to r}^t(i)}{\sum_j \alpha_t^r(j)\beta_t^r(j)\prod_q \mu_{q\to r}^t(j)}$$

For proximity signals, where Z is the normalization constant

$$\mu_{r\to q}^t(i) = \frac{1}{Z}\sum_{s_t^r} \left[f_{rq}^t(s_t^r, s_t^q = i)\alpha_t^r(i)\beta_t^r(i) \prod_{p\neq q} \mu_{p\to r}^t(i)\right] \quad \forall q$$

Initially, all messages are set to one. Note that in our problem setting, the messages that are being passed from one robot to another can be interpreted as

a local marginal belief about the state distribution that different robots have. LBP essentially tries to insure the consistency of these different local beliefs at the times when proximity signals are observed.

We iterate through robots $r = 1, 2, ..., R$ sequentially, running the above inference algorithm. When completed, the algorithm has computed $\gamma_t^r(i)$ and all outgoing messages $\mu_{r\to q}^t$ are sent. We monitor the convergence of this LBP by the absolute difference between successive marginal beliefs and continue passing messages until these stabilize (which typically takes 15-20 iterations) or until a maximum number of iterations, which we set to 25, has been reached. After inference has converged, we perform an M-step to update the model parameters Θ and then repeat the iterative inference procedure before updating the parameters again.

The final learning and inference algorithm is given below.

Learning Algorithm for Multiple Interacting Robots :

- Repeat until parameters converged or maximum number of learning iterations
  - E-step: Perform Inference Step (see above box)
    - While inference not yet converged and below maximum inference iterations
      - Run modified FB recursion for each robot $r = 1, ..., R$, which
        - takes into account the incoming messages from other robots
        - computes marginal beliefs about robot's state occupation
        - sends appropriates messages to other robots.
    - If inference does not converge, Run standard FB on disconnected state chains.
    - Compute marginal beliefs
  - Perform an M-step to update parameters

In rare cases, when LBP fails to converge, we may make some further approximations. (However, in all of the examples presented here this never occurred.) It is possible to employ "damped" versions of LBP [8] which often converge empirically, or resort to more tedious and slow double-loop algorithms that are always guarantee to converge [6]. However, it is generally believed that the accuracy of the answers returned by these damped or double-loop algorithms in cases where regular LBP has trouble converging may be quite poor. In more complex examples when LBP failed to converge, we simply ignore all robot interactions for one iteration, and just run the standard forward-backward (FB) recursions on the disconnected state chains in parallel. We then perform an M-step as usual and return to loopy propagation at the next inference step.

6 Experimental Results

We experimented with single and multiple robots in a 15x15 grid world using simulated logs from 3 continuous valued and 1 binary valued sensor. The functions defining the 3 continuous sensors were generated at random using mixtures of small numbers of Gaussians. The binary sensor measures contact with the wall

(world edge), using the output model $P(y_{wall} = 1|s_t = i) = 1 - \epsilon$ if i is a wall state and zero if i is not a wall state. In our simulations, $\epsilon = 0.1$ Smooth, continuous trajectories of nonconstant velocity were generated and sampled at regular time intervals. The values of the 3 sensors at these continuous positions and discrete times was corrupted with Gaussian noise of standard deviation 0.1, with the scale of sensor readings being from 0 to 1. (In our experiments we assumed that the output model for the binary sensor only was known to the robot: in effect this lets the robot guess when it has reached the limits of the region it is exploring, although it does not know which of the four wall it might be contacting.)

Figure 7 (top panel) shows the true sensor maps for the continuous sensors, along with a subsequence of one continuous trajectory. It also shows the state discretization of the world as dotted lines. In total, we generated 4 sequences of 2500 noisy observations where each observation consisted of 3 continuous valued and 1 binary valued sensor (fig 1).

Figure 7 (second from top panel) shows the results of applying our SLAS algorithm assuming that these 4 sequences were generated by 4 separate excursions by a single robot (or 4 excursions by non-interacting robots). The reconstructed maps have been flipped vertically for the display, since of course the algorithm cannot recover absolute orientation. (Of course the sensor maps we estimate are piecewise constant at the scale of the grid resolution; but graphically some smoothing has also taken place when drawing the contour lines.)

The average RMS localization error between our reconstructed trajectories (computed using Viterbi decoding in this case) and the true trajectories is only 1.02 times the grid size (averaged over all reconstructed trajectories). This implies that, on average, we can estimate the agents' locations to within our discretization error limit.

Figure 7 (middle panel) shows the results of applying our multi-SLAS algorithm, assuming that the 4 trajectories were executed in two excursions by two interacting robots. Notice that the same total amount of data is used, except for the inclusion of the proximity sensors (in fact exactly the same data traces are used, we just pretend they came from two robots instead of four). The proximity signals were generated with probability $1 - \delta$ if the true (continuous) positions of the robots were within a distance of 1.0 grid units and with probability zero otherwise. (Notice that this is a slightly different process than the one assumed by the robots during inference.) In our experiments, $\delta = 0.1$. This resulted in proximity signals being observed at 3-5% of timesteps for each agent on average. Keep in mind that the proximity signals for each robot are noisy and this noise is independent; this means that at time t robot p may detect robot q but not vice versa.

We can see that by trying to enforce consistency between the robots using LBP, our approximate algorithm improves the survey map as well as the trajectory reconstructions as compared to exact inference without interaction signals. (The RMS position error in this case went down to 0.81.)

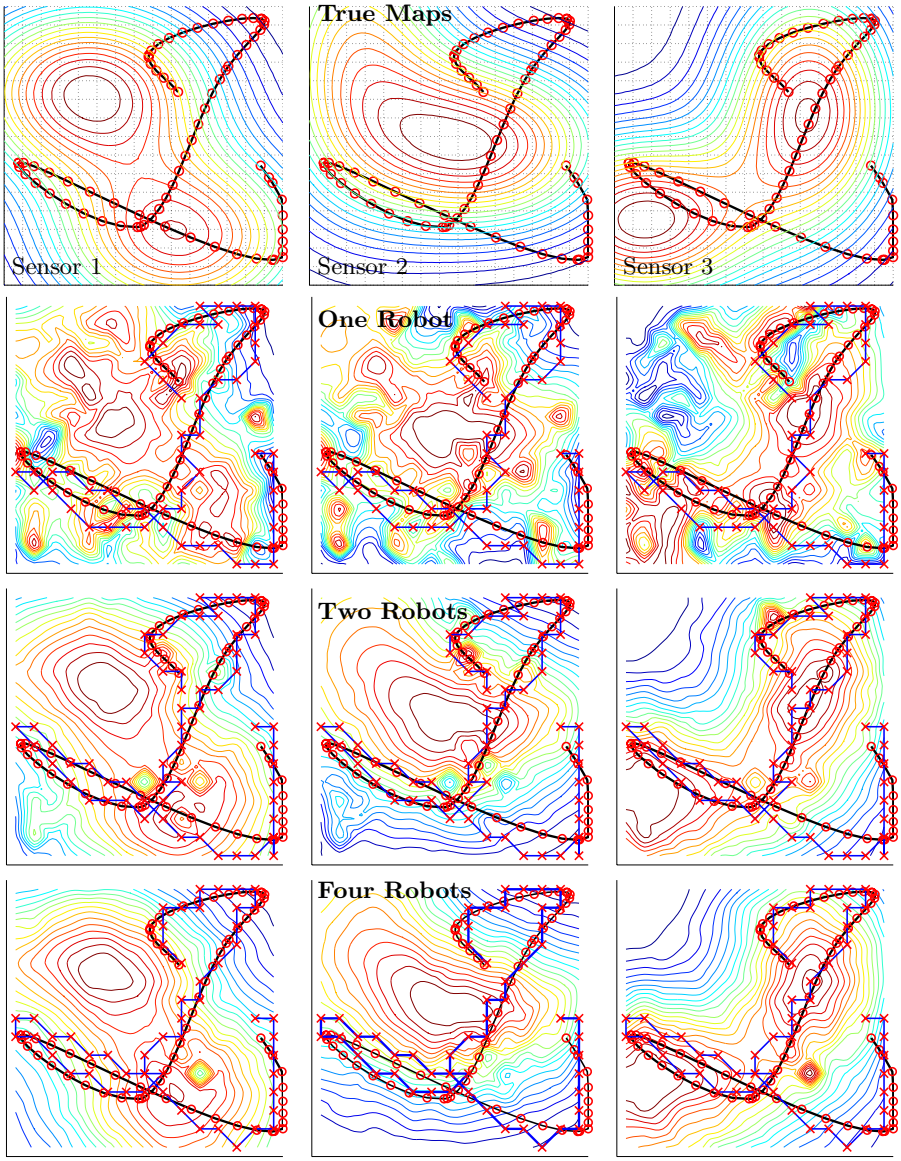

Fig. 7. Display of the true three (left to right) maps of sensor readings along with the part of the true trajectory taken by the robot (top panel), reconstructed maps with inferred trajectories for one (second), two (third) and four (bottom) robots. The "o"s indicate the time slices when the true observation was taken, and "x"s indicate the inferred path taken by the robot.

Finally, Figure 7 (bottom panel) shows the results of running our multi-SLAS algorithm assuming four robots navigated the space simultaneously. The pairwise proximity signals were generated as above. In this case, the survey and trajectory reconstructions have further improved. (RMS position error is 0.76.) Because there were more agents, the graph in this case contains proximity signals at 10-12% of the timesteps.

7 Discussion & Conclusions

In this paper, we have presented a variety of algorithms for solving the simultaneous surveying and localization problem when an unknown environment is explored by multiple interacting agents. Although a simple discretization of the world leads to a tractable constrained HMM architecture in the case of a single agent, multiple interacting agents cause exact learning and inference to become exponentially expensive. Rather than ignoring interactions, we have derived an efficient approximate multi-agent inference algorithm for this architecture based on Loopy Belief Propagation. Although our algorithm does not perform exact inference, we have shown on simple grid world experiments that its performance – both in terms of survey parameters and localization – is superior to performing exact inference while ignoring agent interactions. Other approximate inference methods, especially those based on applying particle filters [1], have been applied to mobile robotics, but this work has focused on mapping occupancy grids (SLAM), and on the single agent setting.

One particularly intriguing byproduct of our learning algorithm is that the intermediate state marginals $\Gamma(i) = \sum_{r=1}^{R} \sum_{t=1}^{T} \gamma_t^r(i)$ contain the estimated occupancy numbers for each grid state. These estimates can potentially be used for traditional mapping (SLAM): states with very low occupation numbers likely correspond to inaccessible regions. Also, these values could be returned as feedback signals to the control algorithm driving the robots to indicate which areas of the world need to be explored further, in the consensus opinion of all agents. Figure 8 displays this statistic for single-robot, two-robot and four-robot experiments using the same data as in the other experiments. It is interesting to note that the map and trajectory reconstruction are inaccurate precisely in areas where we think we are most uncertain about the world according to Γ.

For localization with a single agent, we have also investigated a continuous trajectory representation which avoids the need to discretize the world. In this setting we have successfully employed the embedded HMM architecture as an optimizer and found that it achieves excellent localization results avoiding both the computational cost and the discretization performance limit of our constrained HMMs.

We are currently developing SLAS algorithms based on the discretization-free continuous representation and using the embedded HMM optimizer. We are particularly interested in possible extensions to the multiple robots case both for localization and for multi-SLAS. Ultimately, we hope to apply these algorithms to real data from teams of mobile agents, for example planetary rovers.

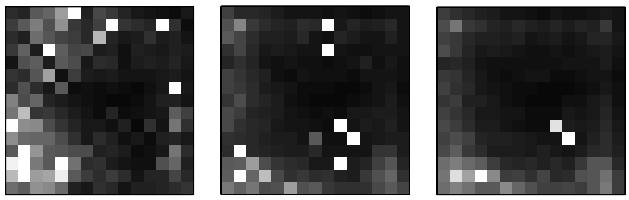

Fig. 8. Display of uncertainty about the current state distribution for 1 robot (left), 2 robots (middle), and 4 robots(right). The uncertainty is measured by Γ. For visualization purposes we display $1/\Gamma$, so white cells correspond to the states which robots are most uncertain about.

Acknowledgments

We thank Tim Barfoot, Martin Wainwright and Max Welling for useful discussions. STR & RRS are supported in part by the Learn Project of IRIS Canada and by NSERC.

References

[1] D. Fox, S. Thrun, W. Burgard, and F. Dellaert. Particle filters for mobile robot localization. In *Sequential Monte Carlo Mehods in Practice*. Springer, 2001.
[2] W. Freeman, E. Pasztor, and O. Carmichael. Learning low-level vision. In *Int. J. Computer Vision*, 2000.
[3] Brendan J. Frey and David J. C. MacKay. A revolution: Belief propagation in graphs with cycles. In Michael I. Jordan, Michael J. Kearns, and Sara A. Solla, editors, *Advances in Neural Information Processing Systems*, volume 10. The MIT Press, 1998.
[4] Zoubin Ghahramani and Michael I. Jordan. Factorial Hidden Markov Models. *Machine Learning*, 29:245–273, January 1996.
[5] W. R. Gilks, S. Richardson, and D. J. Spiegelhalter. *Markov Chain Monte Carlo in Practice*. Chapman_Hall, 1996.
[6] Tom Heskes. Stable fixed points of loopy belief propagation are minima of the bethe free energy. In *Advances in Neural Information Processing Systems*, volume 15, 2003.
[7] Frank R. Kschischang, Brendan J. Frey, and Hans-Andrea Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 47(2):498–519, 2001.
[8] Kevin Patrick Murphy, Yair Weiss, and Michael I. Jordan. Loopy Belief Propagation for Approximate Inference: An Empirical Study. In *Proceedings of UAI*, pages 467–475. Morgan Kaufmann Publishers, July 1999.
[9] Radford Neal. Markov chain sampling for non-linear state space models using embedded hidden Markov models. Technical Report 0304, Dept. of Statistics, University of Toronto, April 2003.
[10] Radford Neal, Matthew Beal, and Sam Roweis. Inferring state sequences for non-linear systems with embedded hidden Markov models. In *Advances in Neural Information Processing Systems*, volume 16, 2004.

[11] J. Pearl. *Probabilistic Reasoning in Intelligent Systems.* Morgan-Kaufman, 1988.
[12] S. T. Roweis. Constrained hidden Markov models. In *Advances in Neural Information Processing Systems*, volume 12, pages 782–788, Cambridge, MA, 1999. MIT Press.
[13] Weiss and Freeman. On the optimality of solutions of the max-product belief-propagation algorithm in arbitrary graphs. *IEEETIT: IEEE Transactions on Information Theory*, 47, 2001.